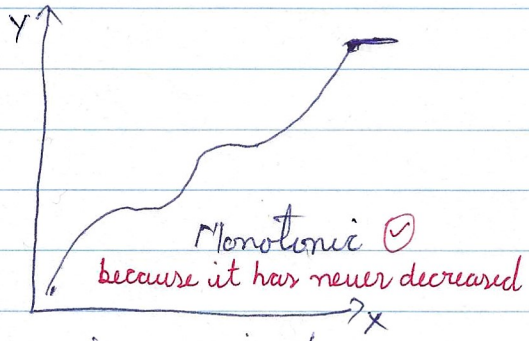
Binary Search Quick Sort and Merge Sort are "divide and conquer" algorithms

BINARY SEARCH 2

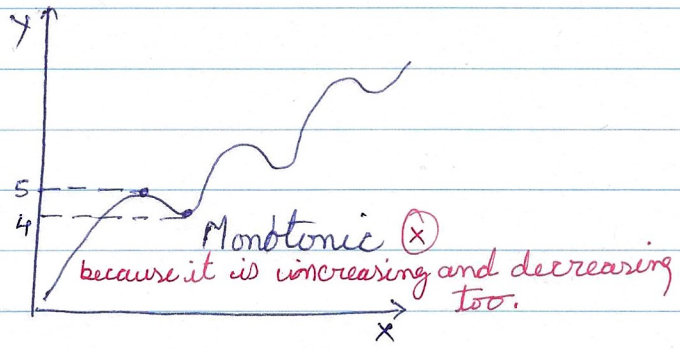
Monotonic function

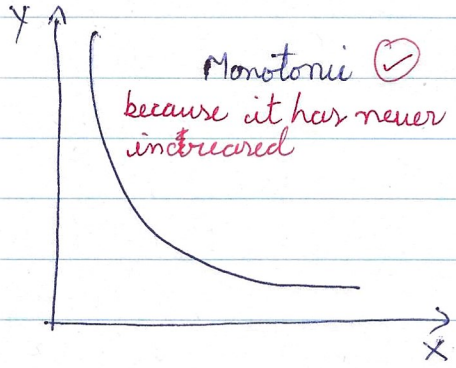
Monotonic function means a function that always follow a single direction. Either it is continuously increasing or continuously decreasing.

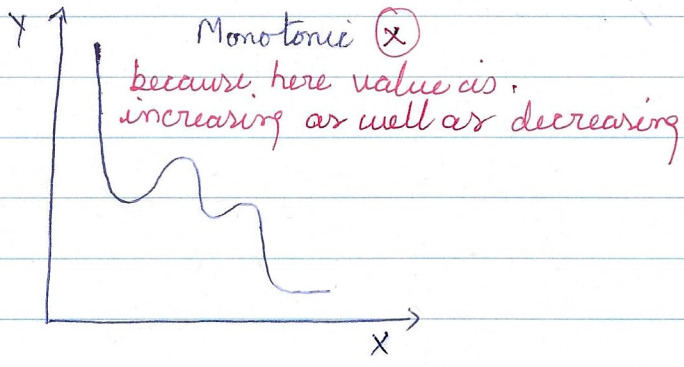
$$y = f(x)$$



increasing / non-decreasing

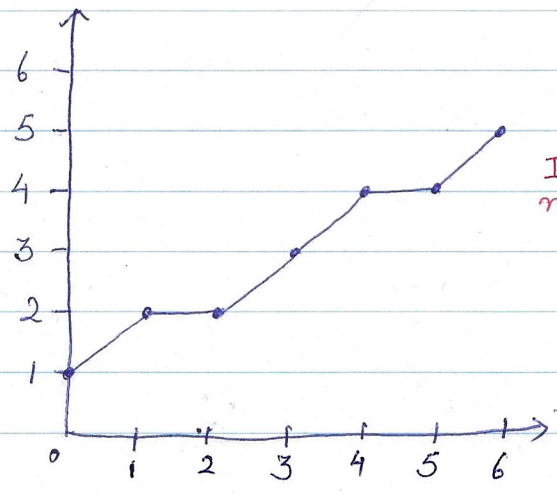






If we plot the graph of the array below on the basis of it's index, will it create a monotonic function?

0	1	2	3	4	5	6
1	2	2	3	4	4	5



Monotonic ✓

It is a non-decreasing monotonic search space

IMPORTANT NOTE: While writing the algorithm for monotonic function with binary search. Always keep in mind, that the function that will run against each mid value of binary search will increase the TC drastically. In maximum cases.

BINARY SEARCH 2

Question 1 What are the various ways of finding square roots of a number? INTEGER PART OF SQUARE ROOT

Approaches

① Linear Search $TC = O(\sqrt{n})$ $SC = O(1)$ To find only integer part of the root not decimal part.

```
N = 50
i = 0
while (i x i <= N) {
    i++;
}
ans = i-1
print i;
```

$O(\sqrt{n}) < O(n)$

if $n = 10^4$

$O(\sqrt{10^4}) < O(10^4)$

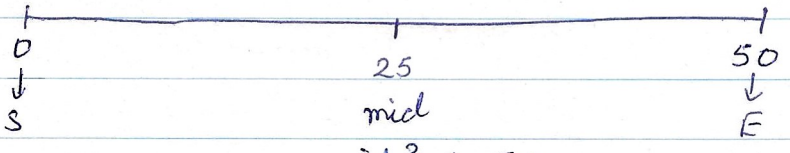
$= O(10^2) < O(10^4)$

$= O(100) < O(10^4)$

② Binary Search $TC = O(\log n)$ $SC = O(1)$

Lets find the square root of a number using binary search.

$N = 50$



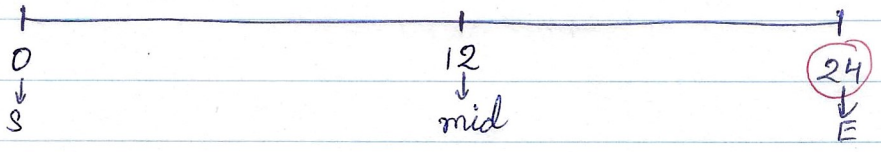
$mid^2 <= 50$

$25 \times 25 <= 50$ ✗

$E = mid - 1$

$= 25 - 1$

$= 24$



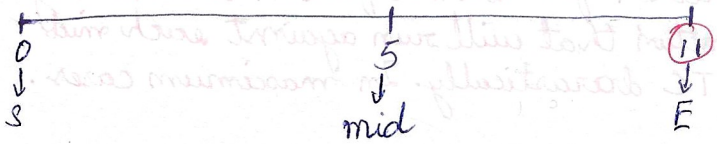
$12 \times 12 <= 50$

$144 <= 50$ ✗

$E = mid - 1$

$= 12 - 1$

$= 11$



N = 50

$mid^2 <= 50$

~~$25 <= 50$~~

$5 \times 5 <= 50$

$25 <= 50$ ✓

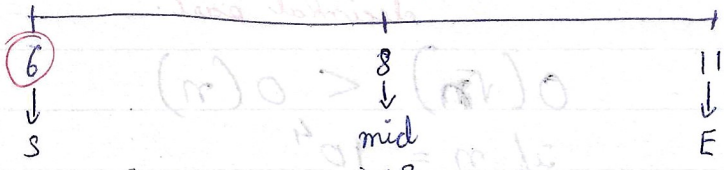
$S = mid + 1$

$S = 5 + 1$

$S = 6$

NOTE: Binary Search reduces the search space by half every time

$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \ldots\ldots 1$



$mid^2 <= N$

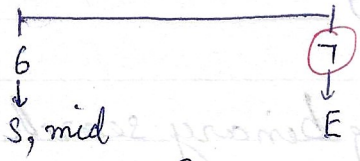
$8 \times 8 <= N$

$64 <= 50$ ✗

$E = mid - 1$

$= 8 - 1$

$= 7$



$mid^2 <= N$

$36 <= 50$ ✓

$S = mid + 1$

$S = 6 + 1$

$= 7$

7

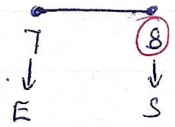
S, E, Mid

$mid^2 <= N$

$49 <= 50$ ✓

$S = mid + 1$

$= 7 + 1 = 8$



S > E Terminate

BINARY SEARCH 2

Question 2 What is the way of finding ~~decimal~~ the value after decimal number while calculating square root of a number? DECIMAL PLACES OF SQUARE ROOT

NOTE: Since we need precision at decimal places we need ~~some~~ Big Do

Approach

(1) Linear Search

Question 1

(a). Fuirst we will calculate the integer part of the answer.

(b) Then we will calculate the numbers on various decimal places one-by-one.

(b.1) We will get ~~this~~ the no. of decimal places needed as part of the question.

(b.2) For each decimal place, we will run an iteration to increase the value of ~~iteration~~ by 1 in each decimal plate iteration. We will ~~have~~ run the iterations till the answer^2 <= N

No. of places after decimal.

```
float SquareRoot ( N, P ) {
    float ans = binSearchInSqRoot (N); // Finding integer part of square root   Question 1
    float inc = 0.1
    for( i = 1; i <= P; i++) {
        while (ans x ans <= N) {
            ans = ans + inc;
        }
        inc = inc / 10;
    }
    return ans;
}
```

BINARY SEARCH 2

Question 3 Farmer John has built a new long farm, with N stalls. The stalls are located along a straight line at positions $x_1, \ldots x_N$.

His C ($2 <= C <= N$) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, Farmer wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

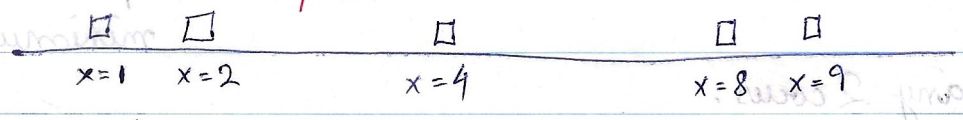
NOTE: Can we apply ~~of~~ binary search to all min-max or max-min problems?

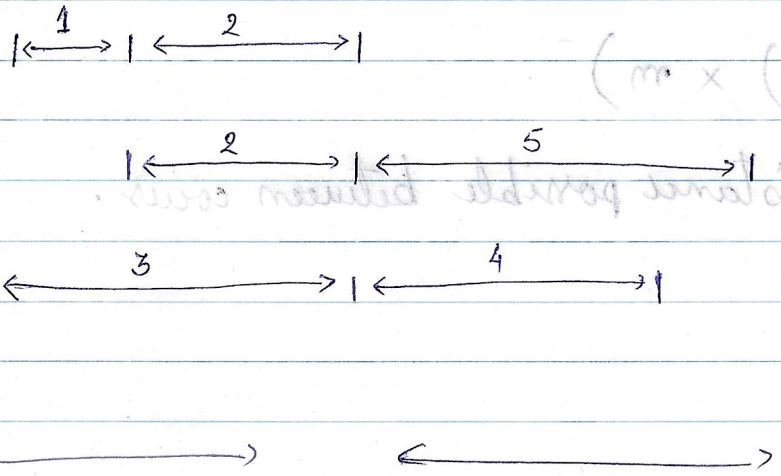
All mini-max or max-min problems could be solved either using binary search or dynamic programming. If problem is monotonic in nature then we will use binary search otherwise dynamic programming

APPROACH 1 Finding all combinations of cows with available stalls.

Explanation N = 5 Stalls , Stalls = [1,4,8,2,9] , C = 3 Cows

Linear Search on search space





dis = min(1,2) = 1

dis = min(2,5) = 2

dis = min(3,4) = 3

In this way we will find all the possible scenarios to maximize the distance

We will create $\boxed{{}^{N}C_{c}}$ combinations to reach an answer which is not optimal.

~~APPROACH 2 → Binary Search~~

$TC = O(^{N}C_{C} \times N)$ // OR // $TC = O(n \times m)$ $SC \rightarrow O(1)$

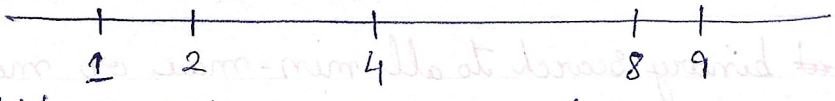
N → No. of Stalls
C → No. of Cows

For each combination we will go through or iterate all the N stalls thats why we multiply ~~all~~ no. of combinations with N.

APPROACH 2 → Binary Search

Monotonic Search Space questions - First of all find out the search space. The search space is the range in which your answer can possibly lie.

(1) In this problem search space is the distance between the stalls



Minimum distance between 2 cows = 1
Maximum distance between 2 cows = 9 - 1 = 8

(2) Now, we can run a binary search between minimum and maximum distance.

(3) We will use one more function / method that will take mid as input and return whether ~~mid~~ we can maintain minimum distance as mid between any 2 cows.

$TC - O(\log(n) \times m)$

n = Maximum distance possible between cows.

m = No. of stalls

BINARY SEARCH 2

Binary Search is an algorithm that can be used to search an element in a ~~sorted dataset~~ monotonic search space.

Search space depends upon the value to be searched.

Depending upon the problem statement we might be optimising for a quantity such as time, distance, cost etc.

- binary search can be applied in all or many optimisation problems if min(max(...)) or max(min(...)) given the problem has a monotonic search space!

BINARY SEARCH 2

Trailing question of 'No. of trailing 0s in N! [N factorial]' NOTES 3

Question 4 Ath Magical Number — You are given 3 positive integers. A, B and C.

Any positive integer is magical if divisible by either B or C.

Return the A^{th} smallest magical number. Since the answer may be very large, return modulo 10^9+7.

Note: Ensure to prevent integer overflow.

Approaches

(1) Linear Search - Starting from 1 to min(B, C) × A we will check the multiples of B and C. We will keep a counter to check the number of magical number i.e. 1st, 4th, 5th or Ath magical number and we will stop when we will reach Ath magical number.

TC - O(min(B, C) × A)

Eg A = 5, B = 2, C = 100
TC example.

(2) Binary Search - Search space is no. of multiples till a particular number. If it matches A we will return it after optimizing it further via binary search.

start = 1 end = min(B, C) × A initial

number of multiples at a particular number N $= \frac{N}{B} + \frac{N}{C} - \frac{N}{LCM(B, C)}$ — (Y)

N will be our mid in the binary search.

$Y \geq A$

start = 1, end = mid - 1

$Y < A$

start = mid + 1, end = ...

In iteration, whenever we will see that the $Y == A$ then we will store NF in a variable called answer and will continue to check if the number smaller than N produces same $Y == A$. If yes, we will store it again in answer variable.

$$TC - O(Log(min(B, C) \times A))$$

BINARY SEARCH 2

Question 5 You are given 2 numbers (~~m~~. n, m); the task is to find the $\sqrt[n]{m}$ (n^{th} root of m)

Answer

Approaches

① **Linear Search** - We will start from no. 1 and keep on moving forward while multiplying it with itself n no. of times. If the result is smaller than m then we move to next number. If result is greater than m, exact n^{th} root of m is not possible. If we find the n^{th} root equal to m, we will return it.

TC - O(M x N)
SC - 1

② **Linear Search with ~~power~~ binary exponentiation method**

Under this approach everything will remain same as first approach except the way of calculating n'th root. Here we will be using ~~power~~ binary exponentiation method. (NOTES 2)
Binary Exponentiation

TC - O(M x $\log_2$(N))
SC - 1

BINARY SEARCH 2

(3) Binary Search

Suppose we need to find $\sqrt[3]{125}$ i.e. $n=3$, $m=125$

→ min = ~~0~~1, max = 125, mid = 63

$mid^n > m$

max = mid − 1

→ min = 1, max = 62, mid = 31

$mid^n > m$

max = mid − 1

→ min = 1, max = 30, mid = 15

$mid^n > m$

max = mid − 1

→ min = 1, max = 14, mid = 7

$mid^n > m$

max = mid − 1

→ min = 1, max = 6, mid = 3

$mid^n < m$

min = mid + 1

→ min = 4, max = 6, mid = 5

$mid^n == m$

return mid;

mid^n

$O(n \times \log m)$ ← (1) mid × mid × mid ... n times

(2) We can also find it using binary exponentiation.

NOTES 2

$O(\log m \times \log n)$ ←

BINARY SEARCH 2

Question 6 Painter's partition problem

Given 2 integers A and B and an array of integers C of size N. Element C[i] represents the length of i^{th} board.

You have to paint all N boards $[C_0, C_1, C_2, C_3 \ldots C_{N-1}]$. There are A painters available and each of them takes B units of time to paint 1 unit of the board.

Calculate and return the minimum time required to paint all boards under the constraints that any painter will only paint contiguous section of the board.

Note:

(1) 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partically by another.

(2) A painter will only paint contiguous boards. This means a configuration where painter 1 paints boards 1 and 3 but not 2 is invalid.

Return the ans % 10000003.

Problem Constraints

$1 <= A <= 1000$

$1 <= B <= 10^6$

$1 <= N <= 10^5$

$1 <= C[i] <= 10^6$

MIN - To find min always take best case like we took in the problem below.
MAX - Max will be total affect of the array provided.

BINARY SEARCH 2

Answer

Approaches

#1 Linear Search.

min PTO

→ ~~minimum time that could be considered is 1 unit.~~

→ max maximum time that could be considered can be calculated by adding total time that could be taken to paint all the boards. [If we had only 1 painter]

For $C = [C_0, C_1, C_2, \ldots, C_{n-1}]$ $\quad B = m$

maximum time = $(C_0 \times m) + (C_1 \times m) + (C_2 \times m) + \ldots + (C_{n-1} \times m)$

From min to max we will check whether painters could paint all the board. The first time value for which painters could paint the boards will be our answer. Iterating from min to max time.

TC - $O(N \times M)$

N → Length of cardboard array.

max M → Maximum time that could be considered.

#2 Binary Search - Run binary search between min time and max time.

A = 2, B = 5, C = [1, 10]

minTime = # #

maxTime

min

Minimum Time Calculation - time taken to paint largest board in the array C,

BINARY SEARCH 2

max(C[i])

minTime = Max length of boards in C × B

= 50 [5 × 10] (B × C[i])

maxTime = (5 × 1) + (5 × 10) = 5 + 50 = 55 (B × C[0] + B × C[1])

→ min = 50, max = 55, mid = 52

P1 (1 × 5) = 5 <= 52 ✓

5 + (5 × 10) = 55 <= 52 ×

P2 (5 × 10) = 50 <= 52 ✓ max = mid - 1

→ min = 50, max = 51, mid = 50

P1 (1 × 5) = 5 <= 50 ✓

5 + (5 × 10) = 55 <= 50 ×

P2 (5 × 10) = 50 <= 50 ✓ max = mid - 1

→ min = 50, max = 49

min <= max ×

return min;

BINARY SEARCH 2

Question 1 Allocate books

Given an array of integers A of size N and an integer B.

The college library has N books. The ith book has A[i] number of pages.

You have to allocate books to B number of students so that the maximum number of pages allocated to a student is minimum.

NOTE:

(1) A book will be allocated to exactly one student. Each student has to be allocated at least one book.

(2) Each student has to be allocated at least one book.

(3) Allotment should be in contiguous order, For eg. A student can not be allocated book 1 and book 3, skipping book 2.

Calculate and return that minimum possible number.

NOTE: Return -1 if a valid assignment is not possible.

Answer

Approaches

(1) Linear Search.

(a) First we will the minimum possible and maximum possible pages that a student or many students can read.

Minimum possible pages - We can find it by assuming that we have same no. of students as no. of books. then MAX (A[0], A[1], ..., A[n-1]) will be our min

BINARY SEARCH 2

Maximum Possible Pages - We can find max pages by assuming that there is only one student who will read all the pages.

MAX = A[0] + A[1] + ... + A[n-2] + A[n-1]

(b)

Once we will get the min and max, we will try allocating books to the students on the ~~basis of~~ contiguous basis for each number N. min <= N <= max

From min to max, first number on which all books could be allocated to all the students will be our answer.

(2) Binary Search.

In binary search we need to run a binary search on min and max that we found while doing linear search. For every mid we will run the function in point (b) of linear search.

A = [12, 34, 67, 90] B = 2

→ min = 90, max = 203, mid = 146

S1 — 12 <= 146 ✓

12 + 34 = 46 <= 146 ✓

12 + 34 + 67 = 113 <= 146 ✓

12 + 34 + 67 + 90 = 203 <= 146 ✗

S2 — 90 <= 146 ✓ max = mid - 1

→ min = 90, max = 145, mid = 117

BINARY SEARCH 2

S1 → 12 <= 117 ✓
12 + 34 = 46 <= 117 ✓
12 + 34 + 67 = 113 <= 117 ✓
12 + 34 + 67 + 90 = 203 <= 117 ✗

S2 → 90 <= 117 ✓ max = mid - 1

→ min = 90, masc = 116, mid = 103

S1 → 12 <= 103 ✓
12 + 34 = 46 <= 103 ✓
12 + 34 + 67 = 113 <= 103 ✗

S2 → 67 <= 103 ✓
67 + 90 = 157 <= 103 ✗ min = mid + 1

→ min = 104, masc = 116, mid = 110

S1 → 12 <= 110 ✓
12 + 34 = 46 <= 110 ✓
12 + 34 + 67 = 113 <= 110 ✗

S2 → 67 <= 110 ✓
67 + 90 = 157 <= 110 ✗ min = mid + 1

→ min = 111, masc = 116, mid = 113

S1 → 12 <= 103 ✓
12 + 34 = 46 <= 113 ✓
12 + 34 + 67 = 113 <= 113 ✓
12 + 34 + 67 + 90 = 203 <= 113 ✗

BINARY SEARCH 2

S2 → 90 <= 113 ✓ masc = mid - 1

→ min = 111, masc = 112, mid = 111

S1 → 12 <= 111 ✓
12 + 34 = 46 <= 111 ✓
12 + 34 + 67 = 113 <= 111 X

S2 → 67 <= 111 ✓
67 + 90 = 157 <= 111 X min = mid + 1

→ min = 112, masc = 112, mid = 112

S1 → 12 <= 112 ✓
12 + 34 = 46 <= 112 ✓
12 + 34 + 67 = 113 <= 112 X

S2 → 67 <= 112 ✓
67 + 90 = 157 <= 112 X min = mid + 1

→ min = 113, masc = 112

min <= masc X

Return min; 113

BINARY SEARCH 2

Question 8 There are N stacks of coins, each stack has certain value of coins.

The stacks are lying on a straight line. You have K friends with whom you can share these coins in K+1 partitions. Your friends are greedy and they will pick the best K parts and you will get the left over one. Maximize the value of coins that you will get.

Answer

Approaches

① Linear Search.

min - Minimum no. of coins that I can get.
masc - Maximum no. of coins that I can get.

In the worst case, I could have stack of coins = no. of friends + me. In that case, I will get the stack that has minimum value.

In best case, I won't have any friends and I will get all the stack of coins.

Now, between min and masc I will run an algorithm over all the numbers. This algorithm will greedely distribute the coins to my friends and leftover will be alloted to me. ~~During~~

During the run we will try to maximize the ~~amount~~ value of coins that I will get.

TC - O(max - min × no. of coin stacks)
SC - O(1)

BINARY SEARCH 2

② Binary Search.

min = calculated in the same way as we did in linear Search
max = ,,

Now, we will own binary search using min and max. Rest of the algorithm will remain same.

TC — O (log (max) × no. of stacks of coins)

Eg 1 coins = [10, 20, 15, 5, 8], K = 3 + 1 = 4

→ min = 5, max = 58, mid = 31

F1 10 >= 31 ×
10 + 20 = 30 >= 31 ×
10 + 20 + 15 = 45 >= 31 ✓

F2 5 >= 31
5 + 8 = 13 >= 31 ×

False max = mid - 1

→ min = 5, max = 30, mid = 17

F1 10 >= 17 ×
10 + 20 = 30 >= 17 ✓

F2 15 >= 17 ×
15 + 5 = 20 >= 17 ✓

F3 8 >= 17 ×
False max = mid - 1

BINARY SEARCH 2

→ min = 5, max = 16, mid = 10

F1 10 >= 10 ✓

F2 20 >= 10 ✓

F3 15 >= 10 ✓

F4 5 >= 10 ×
5 + 8 = 13 >= 10 ✓

True min = mid + 1

→ min = 11, max = 16, mid = 13

F1 10 >= 13 ×
10 + 20 = 30 >= 13 ✓

~~F2 20 >= 1~~

F2 15 >= 13 ✓

F3 5 >= 13 ×
5 + 8 = 13 >= 13 ✓

False max = mid - 1

→ min = 11, max = 12, mid = 11

F1 10 >= 11 ×
10 + 20 = 30 >= 11 ✓

BINARY SEARCH 2

F2 $15 >= 11$ ✓

F3 $5 >= 11$ ×

$5 + 8 = 13 >= 11$ ✓

False $max = mid - 1$

→ $min = 11$, $max = 10$, ~~$mid = 10$~~

$min > max$

return max;

BINARY SEARCH 2

Question 9 Given an array arr[] and an integer K where K is smaller than the size of array, the task is to find the K^{th} smallest element in the given array. It is given that all the array elements are distinct.

Answer

Approaches

① Sorting array and returning $K-1^{th}$ element.
TC - $O(n \log n)$
SC - $O(1)$

② Binary Search

min = Minimum element in the given array.
max = Maximum element in the given array.

Now we will run binary search between min and max. For every mid we will find the no. of elements <= mid and max of that elements should be equal to mid.

ⓐ (no. of elements <= mid) == K AND
(max(no. of elements) == mid)

Return mid.

ⓑ no. of elements < K

min = mid + 1

ⓒ Else

max = mid - 1

BINARY SEARCH 2

SC - O(1)

TC - O(n x log(max element - min element))

NOTE: If max - min < no. of elements in the array, then Binary Search is better otherwise sorting is better to find answer.

(3) Quick Select

Quick Select is an algorithm which is partially based on Quick Sort algorithm. Quick Select uses the "partition" method of Quick Sort algorithm.

Eg: arr = [3, 9, 8, 6, 5, 4, 19, 33, 7] , K = 3 , N = 9

(indices: 0 1 2 3 4 5 6 7 8)

L = 0, E = N-1 = 8, i = 0, j = 0, pivot = arr[E], pivotIdx = E

arr = [3, 9, 8, 6, 5, 4, 19, 33, 7]

i, j → 3 ; i → 9 ; Pivot → 7

3 < 7 i.e. arr[j] <= pivot

arr = [3, 9, 8, 6, 5, 4, 19, 33, 7]

i → 9 ; j → 9 ; Pivot → 7

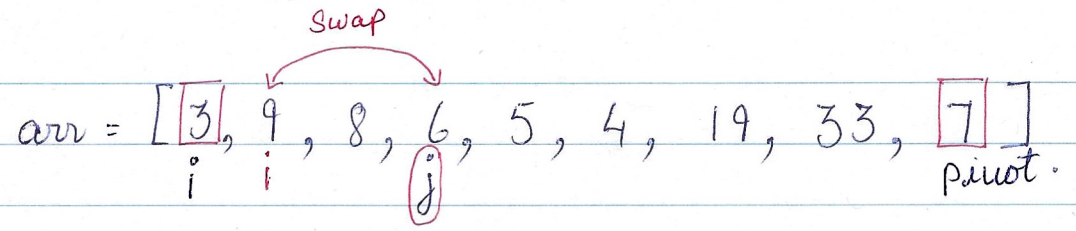
9 > 7 i.e. arr[j] > pivot.

arr = [3, 9, 8, 6, 5, 4, 19, 33, 7]

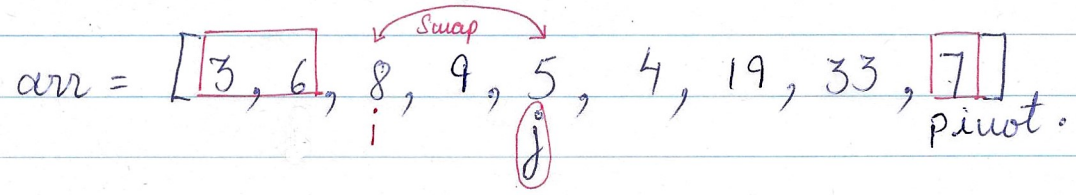
i → 9 ; j → 8 ; Pivot → 7

8 > 7 i.e. arr[j] > pivot.

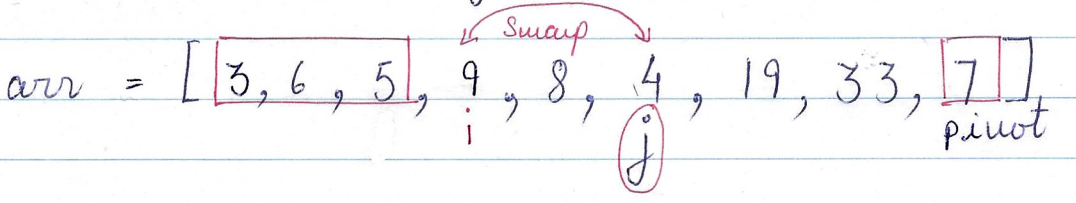
BINARY SEARCH 2



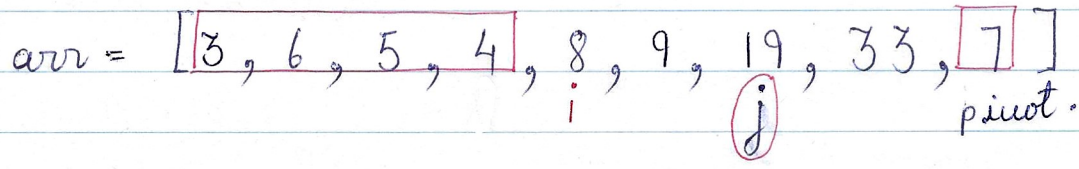
6 <= 7, arr[j] <= Pivot



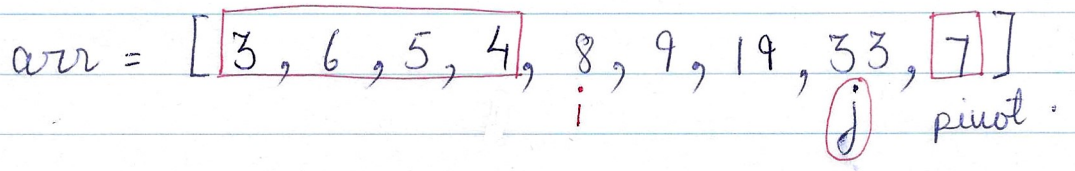
5 <= 7, arr[j] <= pivot.



4 <= 7, arr[j] <= pivot.

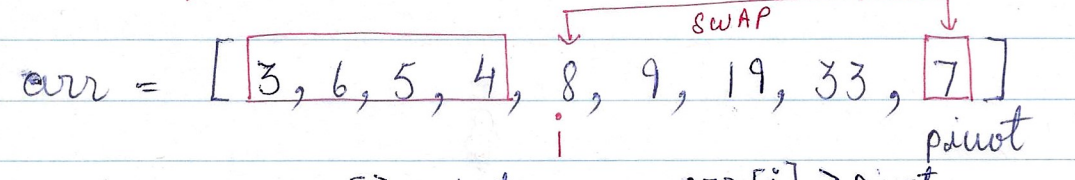


19 > 7, arr[j] > pivot.

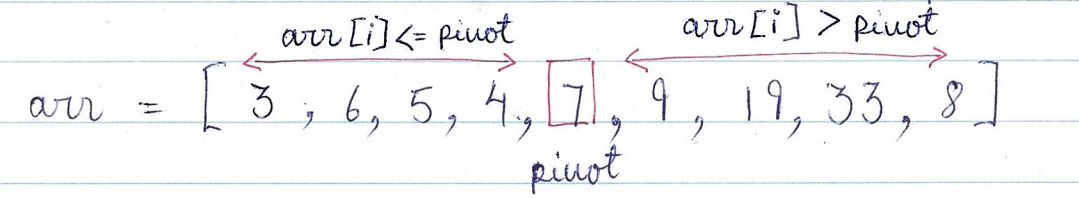


33 > 7, arr[j] > pivot.

Iteration finished here. Now we will swap pivot with ith element.



pivot Idx = 4



BINARY SEARCH 2

pivotIdx > K-1 *

3

L = 0, E = pivotIdx - 1, i = 0, j = 0, pivotIdx = E, pivot = arr[E]

3

4

	0	1	2	3		
arr = [	3	, 6	, 5	, 4	, ...	, 8]
	i, j			pivot		

3 <= 4, arr[j] <= pivot

arr = [	3	, 6	, 5	, 4	, ...	, 8]
		j i		pivot		

6 > 4, arr[j] > pivot

arr = [	3	, 6	, 5	, 4	, ...	, 8]
		i	j	pivot		

Iteration finished here. Now, we will swap pivot with ith element.

		swap				
arr = [	3	, 6	, 5	, 4	, ...	, 8]
		i		pivot		

pivotIdx = 1

arr = [	3	, 4	, 5	, 6	, ...	, 8]
		pivot				

arr[i] <= pivot arr[i] > pivot

*

pivotIdx < K-1

1 < 2

BINARY SEARCH 2

L = 2 = pivotIdx + 1, E = 3, i = 2, j = 2, pivotIdx = 3, pivot = 6 (arr[E])

arr = [..., 5, 6, ..., 8]
i, j — pivot

5 <= 6, arr[j] <= pivot

Iteration finished here. Now, we will swap pivot with ith element.

Swap

arr = [..., 5, 6, ..., 8]
i pivot

pivotIdx = 3

pivotIdx > K - 1
3 > 2

L = 2, E = pivotIdx - 1 = 2, i = 2, j = 2, pivotIdx = E = 2, pivot = arr[E] = 5

arr = [..., 5, ...]
i, j, pivot

pivotIdx = 2

Iteration finished here.

pivotIdx == K - 1
2 == 2

Ans arr[2] = 5 Kth Smallest Element

BINARY SEARCH 2

Time Complexity Calculation – Average Case under assumption that pivot will always fall in middle. Quick Select algo.

Time of select an element in array of size 'n'.

For best and average cases

$$T(n) = \underbrace{C \cdot n}_{\text{constant}} + T(K-1)$$

$C \cdot n$ → TC of partition function

$T(K-1)$ → TC of rest of quick select algorithm.

$T(K-1)$ – Since K is not deterministic. But we can assume the pivot will be in middle always.

$T(K-1)$ will become $T(n/2)$ in above equation.

Equation Method – For recursive solution

$$T(n) = C \cdot n + T(n/2)$$

$$T(n/2) = C \cdot n/2 + T(n/4)$$

$$T(n/4) = C \cdot n/4 + T(n/8)$$

$$\vdots$$

$$T(1) = C \cdot 1 + 0$$

$$T(n) = C\left(n + \frac{n}{2} + \frac{n}{4} + \cdots + 1\right)$$

$$= Cn\underbrace{\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{n}\right)}_{2}$$

$$= 2Cn \Rightarrow O(2Cn) = \boxed{O(n)}$$

(2C: constant)

BINARY SEARCH 2

Quick Select - Time Complexity for worst case

[1, 2, 3, 4, (5)] K = 1

[1, 2, 3, (4)] ← pivot

[1, 2, (3)] ← pivot

[1, (2)] ← pivot

[(1)] ← pivot

, Find 1st smallest element in already sorted Array.
If array is already sorted then it will produce the worst case in the way shown on the side. In each iteration, we will only ~~get~~ remove one number from the search space. which will make the process almost linear.

Equation Method – For Recursive Solution.

$$T(n) = \overset{\text{constant}}{c}n + T(n-1)$$

$$T(n-1) = c(n-1) + T(n-2)$$

$$T(n-2) = c(n-2) + T(n-3)$$

$$\vdots$$

$$T(1) = c(1) + 0$$

$$T(n) = c(n + (n-1) + (n-2) + \dots + 1)$$

$$= cn^2 \Rightarrow O(\underset{\text{constant}}{c}n^2) = O(n^2)$$

probability of hitting the worst case	=	$\frac{1}{n!}$

Because there are only n! ways in which elements of an array could be placed